

1. Class and Instance

```
In [1]: #define a class  
class Employee:  
    pass
```

```
In [2]: emp_1 = Employee()  
        emp_2 = Employee()
```

```
In [3]: print(emp_1)  
        print(emp_2)
```

```
<__main__.Employee object at 0x7efdbc11ecc0>  
<__main__.Employee object at 0x7efdbc11ec88>
```

```
In [4]: #making an instance manually  
        #here emp_1 is an instance  
        emp_1.first = 'Rahatuzzaman'  
        emp_1.last = 'Roni'  
        emp_1.email = 'Rahatuzzaman.Roni@gmail.com'  
        emp_1.pay = 50000  
  
        emp_2.first = 'Test'  
        emp_2.last = 'User'  
        emp_2.email = 'Test.User@gmail.com'  
        emp_2.pay = 10000
```

```
In [5]: print(emp_1.email)  
        print(emp_2.email)
```

```
Rahatuzzaman.Roni@gmail.com  
Test.User@gmail.com
```

It is not so useful. So we have to create an init method in the class

init is a method (constructor) in the Employee class first is an instance variable in self.first

instance variable is declared by self.something

```
In [6]: class Employee:  
        def __init__(self, first, last, pay):  
            self.first = first  
            self.last = last  
            self.pay = pay  
            self.email = first + '.' + last + '@gmail.com'
```

```
In [7]: emp_1 = Employee('Rahatuzzaman','Roni',50000)
emp_2 = Employee('Test','User',10000)
```

emp_1 is passed in as self

Now we should add all the attribute(first,last,pay) with emp_1

```
In [8]: print(emp_1)
print(emp_2)
```

```
<__main__.Employee object at 0x7efdbc0d2160>
<__main__.Employee object at 0x7efdbc0d2128>
```

```
In [9]: print(emp_1.email)
print(emp_2.last)
```

```
Rahatuzzaman.Roni@gmail.com
User
```

```
In [10]: print('{} {}'.format(emp_1.first,emp_1.last))
```

```
Rahatuzzaman Roni
```

Now I will creat a method called fullname

Each method with in a class Automatically takes the instance as the first argument and we are going to always call that ****self****

```
In [11]: class Employee:
def __init__(self,first,last,pay):
    self.first = first
    self.last = last
    self.pay = pay
    self.email = first + '.' + last + '@gmail.com'

def fullname(self):
    return '{} {}'.format(self.first,self.last)
```

As fullname() is a methode so it should be write with paranthesis.

if we write without paranthesis then it print method without return the value

```
In [12]: emp_1 = Employee('Rahatuzzaman','Roni',50000)
emp_2 = Employee('Test','User',10000)
print(emp_1.email)
print(emp_2.email)
print(emp_1.fullname())
print(emp_2.fullname())
```

Rahatuzzaman.Roni@gmail.com
Test.User@gmail.com
Rahatuzzaman Roni
Test User

```
In [13]: print(emp_1.fullname()) #instance.method()
print(Employee.fullname(emp_1)) #class.method(instance)
```

Rahatuzzaman Roni
Rahatuzzaman Roni

2. Class Variables

```
In [14]: class Employee:
def __init__(self,first,last,pay):
    self.first = first
    self.last = last
    self.pay = pay
    self.email = first + '.' + last + '@gmail.com'

def fullname(self):
    return '{} {}'.format(self.first,self.last)
def apply_raise(self):
    self.pay = int(self.pay * 1.04)
```

```
In [15]: emp_1 = Employee('Rahatuzzaman','Roni',50000)
emp_2 = Employee('Test','User',10000)
```

```
In [16]: print(emp_1.pay)
emp_1.apply_raise()
print(emp_1.pay)
print(emp_1.apply_raise)
```

50000
52000
<bound method Employee.apply_raise of <__main__.Employee object at 0x7efdbc0d2390>>

```
In [17]: class Employee:
        raise_amount = 1.04
        def __init__(self,first,last,pay):
            self.first = first
            self.last = last
            self.pay = pay
            self.email = first + '.' + last + '@gmail.com'

        def fullname(self):
            return '{} {}'.format(self.first,self.last)
        def apply_raise(self):
            self.pay = int(self.pay * self.raise_amount)
```

```
In [18]: emp_1 = Employee('Rahatuzzaman','Roni',50000)
        emp_2 = Employee('Test','User',10000)
```

```
In [19]: print(emp_1.raise_amount)
        print(Employee.raise_amount)
        print(emp_2.raise_amount)
```

```
1.04
1.04
1.04
```

```
In [20]: print(emp_1.__dict__)
```

```
{'first': 'Rahatuzzaman', 'last': 'Roni', 'pay': 50000, 'email': 'Rahatuzzaman.Roni@gmail.com'}
```

```
In [21]: print(Employee.__dict__)
```

```
{'__module__': '__main__', 'raise_amount': 1.04, '__init__': <function Employee.__init__ at 0x7efdbc0cdc80>, 'fullname': <function Employee.fullname at 0x7efdbc0cdd08>, 'apply_raise': <function Employee.apply_raise at 0x7efdbc0cdd90>, '__dict__': <attribute '__dict__' of 'Employee' objects>, '__weakref__': <attribute '__weakref__' of 'Employee' objects>, '__doc__': None}
```

```
In [22]: Employee.raise_amount = 1.05
        print(emp_1.raise_amount)
        print(Employee.raise_amount)
        print(emp_2.raise_amount)
```

```
1.05
1.05
1.05
```

```
In [23]: emp_1.raise_amount = 1.02
print(emp_1.raise_amount)
print(Employee.raise_amount)
print(emp_2.raise_amount)
```

```
1.02
1.05
1.05
```

```
In [24]: print(emp_1.__dict__)
```

```
{'first': 'Rahatuzzaman', 'last': 'Roni', 'pay': 50000, 'email': 'Rahatuzzaman.Roni@gmail.com',
'raise_amount': 1.02}
```

```
In [25]: class Employee:
raise_amount = 1.04
num_emps = 0
def __init__(self,first,last,pay):
    self.first = first
    self.last = last
    self.pay = pay
    self.email = first + '.' + last + '@gmail.com'
    Employee.num_emps += 1

def fullname(self):
    return '{} {}'.format(self.first,self.last)
def apply_raise(self):
    self.pay = int(self.pay * self.raise_amount)
```

First of all I still don't set any of employee. Let's check that

```
In [26]: print(Employee.num_emps)
```

```
0
```

Now I am going to create two employees and check it out

```
In [27]: emp_1 = Employee('Rahatuzzaman','Roni',50000)
emp_2 = Employee('Test','User',10000)
```

```
In [28]: print(Employee.num_emps)
```

```
2
```

Now it shows that the number of employees are 2

It is the difference between instance variable and class variables

3. classmethods and staticmethods

```
In [29]: class Employee:

    num_of_emps = 0
    raise_amt = 1.04

    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.email = first + '.' + last + '@email.com'
        self.pay = pay

    Employee.num_of_emps += 1

    def fullname(self):
        return '{} {}'.format(self.first, self.last)

    def apply_raise(self):
        self.pay = int(self.pay * self.raise_amt)

    @classmethod
    def set_raise_amt(cls, amount):
        cls.raise_amt = amount

    @classmethod
    def from_string(cls, emp_str):
        first, last, pay = emp_str.split('-')
        return cls(first, last, pay)
```

```
In [30]: emp_str_1 = 'Rahatuzzaman-Roni-70000'
emp_str_2 = 'Steve-Smith-30000'
emp_str_3 = 'Jane-Doe-90000'
```

```
In [31]: new_emp_1 = Employee.from_string(emp_str_1)
```

```
In [32]: print(new_emp_1.email)
print(new_emp_1.pay)
```

```
Rahatuzzaman.Roni@email.com
70000
```

```
In [33]: class Employee:

num_of_emps = 0
raise_amt = 1.04

def __init__(self, first, last, pay):
    self.first = first
    self.last = last
    self.email = first + '.' + last + '@email.com'
    self.pay = pay

    Employee.num_of_emps += 1

def fullname(self):
    return '{} {}'.format(self.first, self.last)

def apply_raise(self):
    self.pay = int(self.pay * self.raise_amt)

@classmethod
def set_raise_amt(cls, amount):
    cls.raise_amt = amount

@classmethod
def from_string(cls, emp_str):
    first, last, pay = emp_str.split('-')
    return cls(first, last, pay)

@staticmethod
def is_workday(day):
    if day.weekday() == 5 or day.weekday() == 6:
        return False
    return True
```

```
In [34]: import datetime
my_date = datetime.date(2020,11,10)
print(Employee.is_workday(my_date))
```

True

4. Inheritance - Creating Subclasses

```
In [35]: class Employee:
        raise_amount = 1.04
        num_emps = 0
        def __init__(self,first,last,pay):
            self.first = first
            self.last = last
            self.pay = pay
            self.email = first + '.' + last + '@gmail.com'
            Employee.num_emps += 1

        def fullname(self):
            return '{} {}'.format(self.first,self.last)
        def apply_raise(self):
            self.pay = int(self.pay * self.raise_amount)

class Developer(Employee):
    raise_amount = 1.10
```

```
In [36]: dev_1 = Employee('Rahatuzzaman','Roni',50000)
        dev_2 = Employee('Test','User',10000)
```

```
In [37]: print(dev_1.email)
        print(dev_2.email)
```

Rahatuzzaman.Roni@gmail.com
Test.User@gmail.com

```
In [38]: dev_1 = Developer('Rahatuzzaman','Roni',50000)
        dev_2 = Developer('Test','User',10000)
```

```
In [39]: print(dev_1.email)
        print(dev_2.email)
```

Rahatuzzaman.Roni@gmail.com
Test.User@gmail.com

In [40]: `print(help(Developer))`

Help on class Developer in module __main__:

```
class Developer(Employee)
|   Method resolution order:
|       Developer
|       Employee
|       builtins.object
|
|   Data and other attributes defined here:
|
|   raise_amount = 1.1
|
|   -----
|   Methods inherited from Employee:
|
|   __init__(self, first, last, pay)
|       Initialize self. See help(type(self)) for accurate signature.
|
|   apply_raise(self)
|
|   fullname(self)
|
|   -----
|   Data descriptors inherited from Employee:
|
|   __dict__
|       dictionary for instance variables (if defined)
|
|   __weakref__
|       list of weak references to the object (if defined)
|
|   -----
|   Data and other attributes inherited from Employee:
|
|   num_emps = 4
```

None

In [41]: `print(dev_1.pay)`
`dev_1.apply_raise()`
`print(dev_1.pay)`

50000
55000

```
In [42]: class Employee:
        raise_amount = 1.04
        num_emps = 0
        def __init__(self,first,last,pay):
            self.first = first
            self.last = last
            self.pay = pay
            self.email = first + '.' + last + '@gmail.com'
            Employee.num_emps += 1

        def fullname(self):
            return '{} {}'.format(self.first,self.last)
        def apply_raise(self):
            self.pay = int(self.pay * self.raise_amount)

class Developer(Employee):
    raise_amount = 1.10
    def __init__(self,first,last,pay,prog_lang):
        super().__init__(first,last,pay) #same as: Employee.__init__(self,first,last,pay)
        self.prog_lang = prog_lang
```

```
In [43]: dev_1 = Developer('Rahatuzzaman','Roni',50000,'Python')
        dev_2 = Developer('Test','User',10000,'Java')
```

```
In [44]: print(dev_1.email)
        print(dev_1.prog_lang)
```

Rahatuzzaman.Roni@gmail.com
Python

```
In [45]: class Employee:
        raise_amount = 1.04
        num_emps = 0
        def __init__(self,first,last,pay):
            self.first = first
            self.last = last
            self.pay = pay
            self.email = first + '.' + last + '@gmail.com'
            Employee.num_emps += 1

        def fullname(self):
            return '{} {}'.format(self.first,self.last)
        def apply_raise(self):
            self.pay = int(self.pay * self.raise_amount)

class Developer(Employee):
    def __init__(self,first,last,pay,prog_lang):
        super().__init__(first,last,pay) #same as: Employee.__init__(self,first,last,pay)
        self.prog_lang = prog_lang

class Manager(Employee):
    def __init__(self,first,last,pay,employees=None):
        super().__init__(first,last,pay) #same as: Employee.__init__(self,first,last,pay)
        if employees is None:
            self.employees = []
        else:
            self.employees = employees
    def add_emp(self,emp):
        if emp not in self.employees :
            self.employees.append(emp)
    def remove_emp(self,emp):
        if emp in self.employees :
            self.employees.remove(emp)
    def print_emp(self):
        for emp in self.employees:
            print('-->',emp.fullname())
```

```
In [46]: dev_1 = Developer('Rahatuzzaman','Roni',50000,'Python')
        dev_2 = Developer('Test','User',10000,'Java')
```

```
In [47]: mng_1 = Manager('Tony','Stark',100000,[dev_1])
```

```
In [48]: print(mng_1.email)
```

Tony.Stark@gmail.com

```
In [49]: mng_1.print_emp()
```

--> Rahatuzzaman Roni

```
In [50]: mng_1.add_emp(dev_2)
mng_1.print_emp()
```

```
--> Rahatuzzaman Roni
--> Test User
```

```
In [51]: mng_1.remove_emp(dev_2)
mng_1.print_emp()
```

```
--> Rahatuzzaman Roni
```

```
In [52]: #is mng_1 is an instance of Manager?
print(isinstance(mng_1,Manager))
```

```
True
```

```
In [53]: print(isinstance(mng_1,Employee))
```

```
True
```

```
In [54]: print(isinstance(mng_1,Developer))
```

```
False
```

```
In [55]: print(issubclass(Developer,Employee))
```

```
True
```

```
In [56]: print(issubclass(Manager,Employee))
```

```
True
```

```
In [57]: print(issubclass(Manager,Developer))
```

```
False
```

5. Special (Magic/Dunder) Methods

In this Python Object-Oriented Tutorial, we will be learning about special methods. These are also called magic or dunder methods. These methods allow us to emulate built-in types or implement operator overloading. These can be extremely powerful if used correctly. We will start by writing a few special methods of our own and then look at how some of them are used in the Standard Library. Let's get started.

```
In [58]: class Employee:
        raise_amount = 1.04
        num_emps = 0
        def __init__(self,first,last,pay):
            self.first = first
            self.last = last
            self.pay = pay
            self.email = first + '.' + last + '@gmail.com'
            Employee.num_emps += 1

        def fullname(self):
            return '{} {}'.format(self.first,self.last)
        def apply_raise(self):
            self.pay = int(self.pay * self.raise_amount)
        def __repr__(self):
            return "Employee('{}' '{}', {})".format(self.first,self.last,self.pay)
        def __str__(self):
            return "{} - {}".format(self.fullname(),self.email)
```

```
In [59]: emp_1 = Employee('Rahatuzzaman','Roni',50000)
        emp_2 = Employee('Tony','Stark',10000)
```

```
In [60]: print(emp_1)
```

Rahatuzzaman Roni - Rahatuzzaman.Roni@gmail.com

```
In [61]: print(repr(emp_1))
        print(str(emp_1))
```

Employee('Rahatuzzaman' 'Roni' , 50000)
Rahatuzzaman Roni - Rahatuzzaman.Roni@gmail.com

```
In [62]: print(emp_1.__repr__())
        print(emp_1.__str__())
```

Employee('Rahatuzzaman' 'Roni' , 50000)
Rahatuzzaman Roni - Rahatuzzaman.Roni@gmail.com

```
In [63]: print(1+2)
```

3

```
In [64]: print(int.__add__(1,2))
```

3

```
In [65]: print(str.__add__('a','b'))
```

ab

```
In [66]: class Employee:
raise_amount = 1.04
num_emps = 0
def __init__(self,first,last,pay):
    self.first = first
    self.last = last
    self.pay = pay
    self.email = first + '.' + last + '@gmail.com'
    Employee.num_emps += 1

def fullname(self):
    return '{} {}'.format(self.first,self.last)
def apply_raise(self):
    self.pay = int(self.pay * self.raise_amount)
def __repr__(self):
    return "Employee('{}' '{}', {})".format(self.first,self.last,self.pay)
def __str__(self):
    return "{} - {}".format(self.fullname(),self.email)
def __add__(self,other):
    return self.pay + other.pay
```

```
In [67]: emp_1 = Employee('Rahatuzzaman','Roni',50000)
emp_2 = Employee('Tony','Stark',10000)
```

```
In [68]: print(emp_1+emp_2)
```

60000

```
In [69]: print(len('Roni'))
```

4

```
In [70]: print('Roni'.__len__())
```

4

```
In [71]: class Employee:
        raise_amount = 1.04
        num_emps = 0
        def __init__(self,first,last,pay):
            self.first = first
            self.last = last
            self.pay = pay
            self.email = first + '.' + last + '@gmail.com'
            Employee.num_emps += 1

        def fullname(self):
            return '{} {}'.format(self.first,self.last)
        def apply_raise(self):
            self.pay = int(self.pay * self.raise_amount)
        def __repr__(self):
            return "Employee('{}' '{}', {})".format(self.first,self.last,self.pay)
        def __str__(self):
            return "{} - {}".format(self.fullname(),self.email)
        def __add__(self,other):
            return self.pay + other.pay
        def __len__(self):
            return len(self.fullname())
```

```
In [72]: emp_1 = Employee('Rahatuzzaman','Roni',50000)
        emp_2 = Employee('Tony','Stark',10000)
```

```
In [73]: print(len(emp_1))
```

17

Property Decorators - Getters, Setters, and Deleters

In this Python Object-Oriented Tutorial, we will be learning about the property decorator. The property decorator allows us to define Class methods that we can access like attributes. This allows us to implement getters, setters, and deleters. Let's get started.

```
In [74]: class Employee:
        def __init__(self,first,last):
            self.first = first
            self.last = last
            self.email = first + '.' + last + '@gmail.com'
        def fullname(self):
            return self.first + ' ' + self.last
```

In [75]: `emp_1 = Employee('Rahatuzzaman' , 'Roni')`

In [76]: `print(emp_1.fullname())`
`print(emp_1.email)`

Rahatuzzaman Roni
Rahatuzzaman.Roni@gmail.com

In [77]: `type(emp_1.fullname)`

Out[77]: method

In [78]: `type(emp_1.fullname())`

Out[78]: str

In [79]: `type(emp_1.email)`

Out[79]: str

In [80]:

```
class Employee:
    def __init__(self,first,last):
        self.first = first
        self.last = last
    def fullname(self):
        return self.first + ' ' + self.last
    def email(self):
        return self.first + '.' + self.last+'@gmail.com'
```

In [81]: `emp_1 = Employee('Rahatuzzaman' , 'Roni')`

In [82]: `print(emp_1.email())`

Rahatuzzaman.Roni@gmail.com

Now I will use a @property decorator to avoid using email() as method

In [83]:

```
class Employee:
    def __init__(self,first,last):
        self.first = first
        self.last = last
    def fullname(self):
        return self.first + ' ' + self.last
    @property
    def email(self):
        return self.first + '.' + self.last+'@gmail.com'
```

In [84]: `emp_1 = Employee('Rahatuzzaman' , 'Roni')`

In [85]: `print(emp_1.email)`

Rahatuzzaman.Roni@gmail.com

In [86]: `emp_1.last = 'Stark'`

In [87]: `print(emp_1.email)`

Rahatuzzaman.Stark@gmail.com

In [88]:

```
class Employee:
    def __init__(self,first,last):
        self.first = first
        self.last = last
    @property
    def fullname(self):
        return self.first + ' ' + self.last
    @property
    def email(self):
        return self.first + '.' + self.last+'@gmail.com'
    @fullname.setter
    def fullname(self,name):
        first,last = name.split(' ')
        self.first = first
        self.last = last
```

In [89]: `emp_1 = Employee('Rahatuzzaman' , 'Roni')`

In [90]: `print(emp_1.email)`

Rahatuzzaman.Roni@gmail.com

In [91]: `print(emp_1.fullname)`

Rahatuzzaman Roni

In [92]: `emp_1.fullname = 'Tony Stark'`

In [93]: `print(emp_1.fullname)`

Tony Stark

```
In [94]: class Employee:
def __init__(self,first,last):
    self.first = first
    self.last = last
@property
def fullname(self):
    return self.first + ' ' + self.last
@property
def email(self):
    return self.first + '.' + self.last+'@gmail.com'
@fullname.setter
def fullname(self,name):
    first,last = name.split(' ')
    self.first = first
    self.last = last
@fullname.deleter
def fullname(self):
    print('Delete Name!')
    self.first = None
    self.last = None
```

```
In [95]: emp_1 = Employee('Rahatuzzaman' , 'Roni')
print(emp_1.email)
emp_1.fullname = 'Tony Stark'
print(emp_1.email)
```

Rahatuzzaman.Roni@gmail.com
Tony.Stark@gmail.com

```
In [96]: del emp_1.fullname
```

Delete Name!